Journal of Computer and System Sciences 77 (2011) 621–636

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Performance modelling of Event-Condition-Action rules in P2P networks

George Papamarkos, Alexandra Poulovassilis *, Peter T. Wood

London Knowledge Lab, Birkbeck, University of London, United Kingdom

ARTICLE INFO

Article history:
Received 15 September 2009
Received in revised form 15 December 2009
Available online 25 February 2010

Keywords:
Peer-to-peer networks
Event-Condition-Action rules

ABSTRACT

This paper develops an analytical model for exploring the performance of processing Event-Condition-Action (ECA) rules in P2P environments, adopting as the main performance criterion the mean time required to complete all rule processing resulting from a top-level update submitted to one of the peers in the network. We examine the analytical model's predictions of how this time varies with the network topology, number of peers, number of rules and degree of data replication between peers. We also describe a simulation of such a P2P ECA rule processing system, and the experiments with this show good agreement with the analytical predictions. The main contributions of this paper are the analytical model itself, and the results of the performance study undertaken using the model, which have been confirmed by the results from the simulator.

1. Introduction

P2P networks provide a natural infrastructure for deploying a wide variety of applications where local autonomy can be maintained while benefitting from sharing resources. Such applications may need to be *reactive*, i.e. to be able to detect the occurrence of specific events at a peer and to respond by automatically executing the appropriate distributed application logic. For example, in an e-learning application this may be notifying a user that a learning object of potential interest to them has been added at some peer in the network; in an e-science application it may be updating data held at a peer in response to new experimental data being produced at another peer.

Event-Condition-Action (ECA) rules are one way of providing such functionality. An ECA rule has the general syntax `on` *event* `if` *condition* `do` *actions*. The event part specifies how the rule is *triggered*. The condition part is a query which determines if the system is in a particular state, in which case the rule *fires*. The action part states the actions to be performed if the rule fires. These actions may in turn cause further events to occur, which may in turn cause more ECA rules to fire. We refer the reader to [18,23] for a general discussion of ECA rules in the context of databases, where they are known as 'triggers'. More broadly, ECA rules are also used in workflow management, network management, personalisation and business process modelling.

Examples of P2P systems that employ ECA rules include [13], where ECA rules are used to encode the policies by which peers join a scientific P2P network; [6] and [10], where ECA rules are used to specify data propagation and coordination policies between peer databases; and [16], where ECA rules are used to provide notification and personalisation services over distributed RDF descriptions stored in a P2P network of learning resources.

In such P2P applications, the question arises of whether the infrastructure needed to perform ECA rule processing is sufficiently scalable to make ECA rules a viable approach for deployment in large-scale systems. In response to this question, this paper presents an analytical model that aims to serve as a tool for studying the performance and scalability aspects of P2P ECA rule processing systems. We focus on applications where each peer stores data conforming to some schema — so-

* Corresponding author.
E-mail addresses: g.papamarkos@dcs.bbk.ac.uk (G. Papamarkos), ap@dcs.bbk.ac.uk (A. Poulovassilis), ptw@dcs.bbk.ac.uk (P.T. Wood).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.02.004

called schema-based P2P systems [6,14,21]. Our analytical model is independent of any specific data model, query language or update language. We adopt as the main performance criterion the mean time required to complete all rule processing resulting from an update submitted by an application. We have prioritised the modelling of update response time so far in our work, rather than various types of resource consumption, because this is the major concern in the e-learning and e-science P2P applications that we have identified to date. In particular, in these applications there are likely to be possibly large numbers of simple ECA rules rather than smaller numbers of computationally intensive rules. None-the-less, a similar modelling approach to the one that we present here could be adopted for modelling other performance aspects, such as CPU, I/O and network usage.

Section 2 of the paper gives an overview of related work in ECA rules and P2P systems. Section 3 defines the characteristics of the P2P environments and ECA rules that we model. Section 4 develops our analytical performance model. Section 5 presents experimental results from a study undertaken using this model, examining how response time varies with the network topology, number of peers, number of rules, and degree of data replication between peers. Section 5 describes a simulation of an archetypal P2P ECA rule processing system, and presents analogous experimental results with this simulator, which show good agreement with the analytical results. Section 6 gives our conclusions and a discussion of future work.

2. Related work

In [13], ECA rules are used encode the policies by which peers can join the Edutella [14] schema-based P2P system. In [16] an ECA language for RDF is defined, called RDFTL, and an architecture for supporting RDFTL rule execution in P2P environments. This work was motivated by the requirements of the Self e-Learning Networks (SeLeNe) project, in which users subscribing to a P2P network of learning resources require notification and personalisation services over RDF descriptions of such resources [12], and this functionality is provided by RDFTL rules. Both Edutella and RDFTL allow vertical, horizontal and hybrid data fragmentation, with possible replication between peers.

Piazza [21] supports semantic integration and global querying of heterogenous data distributed over a P2P network. Such work is complementary to our investigation of ECA rule processing aspects. In particular, we assume here that the peer schemas are fragments of a single (notional) global schema, and hence that schema mapping services between peers are not required.

In LRM [6] and Hyperion [10], ECA rules are used in order to encode policies for data propagation and coordination between peer databases. Data coordination may require the reconciliation and integration of data at query time or the maintenance of data consistency among peers. Neither system assumes the presence of a global schema, and peers need to be identified explicitly within ECA rules (in contrast to the ECA rule syntax that we assume here).

In [11], ECA rules are used to implement mobile query agents in P2P networks. Each peer supports a database and ECA rule processing capabilities. The event parts of ECA rules trigger the activation of mobile query processing functionality across the network, while the condition parts identify additional requirements for that functionality to be activated. In [24], ECA rules are used for supporting event monitoring in sensor networks. Event channels are established that are responsible for the distribution of events, following a publish/subscribe model. Simple ECA rules can be stored on sensor nodes, and the actions of these rules can activate actuator nodes which can then trigger alarms.

A P2P model for distributing and replicating Active XML documents is presented in [1]. Location-aware extensions of XPath and XQuery are used. A replication algorithm provides recommendations regarding what and where to replicate data in order to improve each peer's query performance. This model of P2P data replication differs from our approach, in that we do not assume explicit references to peers or locations within ECA rules; instead, in our approach data replication results implicitly from the execution of ECA rule actions at all peers on which these updates can successfully be applied.

Previous work on modelling and performance evaluation of ECA rule processing systems (in centralised systems) has focussed on benchmarking and simulation. The BEAST benchmark is used in [8] to measure the performance of event detection and rule execution. Simulation experiments are performed in [3] to measure the performance trade-offs for different ECA rule execution semantics. The average transaction response time, defined as the time elapsed from the transaction's arrival at the execution queue to its successful completion, is used as the main performance measure. We adopt a similar, response-time oriented, performance measure in our analytical model for P2P ECA rule execution.

Finally, we note that content-based publish/subscribe is an alternative technology to ECA rules for some P2P applications. P2P networks that support publish/subscribe, such as [7,22], are able to support more sophisticated event definition and event detection than is feasible with ECA rules. On the other hand, ECA rules allow the definition and execution of more complex actions than just the simple notifications of publish/subscribe.

3. ECA rules in P2P environments

We assume in this paper a P2P network that comprises a set of *peer* servers and a set of *superpeer* servers. Each peer communicates with just one superpeer, called its supervising peer, and all superpeers can communicate with each other. The set of peers with which a superpeer communicates is termed its peergroup. This kind of P2P network is typical in e-science and e-learning P2P applications [13,14,16]. Moreover, it has a good fit with group-based modes of collaborative working and learning in these domains, in which each person is a member of a group and is interacting with a peer server,

each group is engaged on a particular task which is coordinated at a superpeer server, there is close communication and synchronisation between the members of a group, and there is looser communication between different groups.

Each peer or superpeer hosts data relating to a fragment of an overall global schema — this is a notional schema i.e. it is not materialised and stored at any single location. Our data distribution model allows vertical, horizontal and hybrid fragmentation of the data, with possible data replication between peers.

Each peer and superpeer supports query, storage and update capabilities over its data. Each superpeer supports also a set of services which provide ECA rule processing capabilities and which interact with the data processing capabilities of the peergroup.

The fragment of the global schema stored at any peer may change as a result of changes in the peer's data. Peers notify their supervising superpeer of any changes to their local schema. Each superpeer's schema is maintained as a superset of the peergroup's individual schemas. Each superpeer defines access privileges over the objects in its schema (e.g. read-only, read-write, private) describing the corresponding access level to the instances of each schema object. There may be more fine-grained access privileges on specific data items. These facilities allow a superpeer to specify which information can be shared with other superpeers outside its peergroup.

An ECA rule has the syntax `on` *event* `if` *condition* `do` *actions*. More formally:

Definition 1. An ECA rule r is defined by a quadruple (e, c, a, m), where e is an expression that matches occurrences of data updates at peers or superpeers, c is an expression that returns a value of `true` or `false` when evaluated (i.e. a Boolean query), $a = a_1, \ldots, a_n$ is sequence of expressions that give rise to data updates, and m indicates whether the rule is set-oriented or instance-oriented (see below).

We sometimes refer to e as the *event part*, c as the *condition part*, and a as the *action part* of a rule. We model only atomic events not composite events, although extension of our analytical model and simulation to composite events and composite event detection would be an interesting area of further work.

ECA rules are created at an individual peer or superpeer as a result of the execution of higher-level application logic (not as a result of rule execution). The rules created at any peer are stored in a Rule Base hosted by its supervising superpeer. A rule created at one site of the network might be replicated at other sites as well. More specifically, when a new rule r is created at a peer P, it is sent to P's superpeer for syntactic validation and storage. From there, r is also forwarded to all other superpeers, and a replica of it is stored at those superpeers where it is possible that an update can occur that matches r's event part, i.e. those superpeers that are *e-relevant* to r (see below). Each rule has a globally unique identifier of the form $SP_i.j$, where SP_i is the originating superpeer's identifier and j a unique identifier for the rule within SP_i's rule base.

Rule execution is triggered by an update occurring at a peer or superpeer. In particular:

Definition 2. Let $r = (e, c, a, m)$ be an ECA rule. Rule r *may be triggered* by an update u if u syntactically matches e.

For each type of event E detectable by the system, there is a system-defined variable *delta_E* which is instantiated with information about the changes (i.e. insertions, deletions or updates) that the event has made to the peer's data. These are known as *delta variables* and they can be referenced within the condition and action parts of rules.

In our model, we assume 'semantic' rather than 'syntactic' triggering i.e. a rule is triggered if an update occurs that syntactically matches its event part *and* the associated delta variable is non-empty (it would be straightforward to simplify our model to capture syntactic triggering):

Definition 3. A rule $r = (e, c, a, m)$ is *triggered* by an update u if u syntactically matches e and the set of instantiations of the delta variable associated with e is non-empty.

Rules may be set-oriented or instance-oriented, which indicated by their fourth component m having value S or I — these two types of rules capture the notions of statement-level and row-level triggers in SQL, respectively.

Definition 4. A set-oriented rule $r = (e, c, a, S)$ *fires* if it is triggered and c evaluates to true. An instance-oriented rule $r = (e, c, a, I)$ *fires* if it is triggered and c evaluates to true for some instantiation of the delta variable associated with e.

If a set-oriented rule $r = (e, c, a, S)$ fires, a copy of a is scheduled for execution — we discuss rule execution in Section 3.1 below. If an instance-oriented rule $r = (e, c, a, I)$ fires, a copy of a is scheduled for execution for each instantiation of the delta variable for which c evaluates to true. Note that we assume here 'Immediate' coupling between a rule's event part and its condition part, as is typically the case in ECA rule processing systems (see [18] for a detailed discussion of ECA rule coupling modes).

In general, several rules may fire as a result of some event occurrence. The set of rules stored at each superpeer may be partially ordered by a rule precedence relationship, $\prec$, specified by the rule designer or the system. If two rules r_i and r_j fire and $r_i \prec r_j$ then the updates generated by r_i are executed before those generated by r_j. If r_i and r_j are not related

by $\prec$, then the updates generated by them are executed in an arbitrary order. It is desirable in the latter case that r_i and r_j have been written in such a way that they *commute*, i.e. that the same final state will result when rule execution terminates irrespective of the order of scheduling of r_i and r_j. Similarly, the order of execution of instances of the action part of an instance-oriented rule is arbitrary, so it is desirable that the rule has been written in such a way that the same final state will result when rule execution terminates irrespective of this order. Discussion of techniques for determining the termination and confluence characteristics of sets of ECA rules is beyond the scope of this paper, and we refer the reader to [2,4,5,16] for an indicative set of references.

In our P2P ECA rule execution model here, each particular instance of a rule's action part executes at a single peer. If there is a need to distribute a sequence of updates across a number of peers in reaction to some event, then rather than specifying one rule of the form `on` e `if` c `do` $a_1, \ldots, a_n$, instead n rules $r_1, \ldots, r_n$ could be specified, where each r_i is `on` e `if` c `do` a_i and $r_1 \prec r_2 \prec \cdots \prec r_n$.

We define three types of *relevance* of a rule to a (peer or superpeer) schema:

Definition 5. Let $r = (e, c, a, m)$ be an ECA rule, where $a = a_1, \ldots, a_n$, and let S be a schema. Rule r is *e-relevant* to S if it is possible that an update can occur on S that matches e. Rule r is *c-relevant* to S if some subquery of c can be evaluated on S. Rule r is *a-relevant* to S if each a_i, $1 \leqslant i \leqslant n$, is a-relevant to S, where a_i is a-relevant to S if it is a syntactically valid update on S.

e-relevance determines at which superpeers a rule will be replicated; c-relevance determines which peers may participate in the evaluation of a condition (unlike events and actions, conditions may be evaluated across multiple sites); and a-relevance determines at which peers instances of a rule's action part will be executed.

Definition 6. A peer or superpeer is *e-relevant*, *c-relevant* or *a-relevant* to a rule r if r is e-, c- or a-relevant, respectively, to the peer or superpeer's schema.

Each rule stored in each superpeer's Rule Base is annotated with the IDs of the local peers that are e-relevant, c-relevant and a-relevant to it; and these annotations are kept synchronised with any changes occurring in peers' and superpeers' schemas, and as peers join or leave the network. We refer the reader to Chapter 6 of [17] for details of how this is achieved in the RDFTL system, and this approach can be applied more generally.

3.1. P2P rule execution

Transactions consisting of queries and updates are submitted by applications to peers or superpeers in the network. The execution of an update at a peer may cause events to occur. These events may cause rules to fire, resulting in new updates to be executed locally or remotely, which may in turn cause further rules to fire. In more detail, rule execution proceeds as follows:

- Whenever an update u is executed at a peer P, P notifies its supervising superpeer *SP* by sending it a description of u, including the data items at P that have been affected.
- *SP* determines which rules in its rule base annotated with P's ID may be triggered by u, by comparing syntactically the description of u and the rules' event parts.[1]
- If a rule $r = (e, c, a, m)$ may have been triggered by u, then *SP* constructs from e an *event query* to send to P for evaluation. P sends the result of this query back to *SP*, which compares the result set against the set of data items affected by u that it has already received from P. If the intersection of these two sets is non-empty, then the rule is actually triggered. The result of this intersection is the set of instantiations of the delta variable corresponding to event e arising from update u.
- If a rule r has been triggered, *SP* coordinates the evaluation of r's condition across peers that may be c-relevant to the rule, after generating an instance of the condition for each instantiation of the delta variable in the case of an instance-oriented rule.
  We assume here that rule conditions are evaluated locally within a peergroup. Global rule condition evaluation across multiple superpeers would need to use global P2P query processing techniques. Its effect on our experimental results presented in Section 5 would clearly depend on the complexity and scalability of the query processing algorithms employed. For example, [20] discusses query routing in P2P networks that use the HyperCup topology and shows that schema-based clustering of peers significantly improves query performance. In order to not confuse the performance of global P2P query processing with the ECA rule processing functionality that is the focus of this paper, we make the assumption, both in our analytical model and our simulator, that rule conditions are evaluated within a single peergroup.

[1] Superpeers may maintain appropriate indexes to support this determination in an efficient and scalable manner — see, for example, the discussion of indexes in the RDFTL system [16].

- If r's condition evaluates to true, *SP* sends each instance of r's action part (one instance if r is a set-oriented rule, and one or more instances if r is an instance-oriented rule) to its local peers that are a-relevant to r for execution.
  SP also sends all instances of r's actions part to all other superpeers. All superpeers that are a-relevant to r will consult their schemas and access privileges in order to determine whether the updates they have received should be scheduled and executed on their local peergroup.
  SP does not expect to receive any acknowledgement from the other peers or superpeers about whether they have received and possibly scheduled these updates, i.e. there is 'Detached' coupling between each ECA rule's condition and its action. This coupling mode is a natural choice given the loose nature of the relationships between the nodes of a P2P network. Handling rules with more complex coupling modes between the condition and the action would also be possible, e.g. 'Immediate' or 'Deferred' coupling, depending on an application's requirements (we refer to [18] for detailed discussions of ECA rule coupling modes). These would require higher levels of synchronisation between the network nodes, and handling of issues such as dynamicity of nodes, nested transactions and event expiration. These issues are discussed in Chapter 6 of [17] in the case of the RDFTL system. Extending our analytical model to capture such aspects would be an interesting area of future work.

4. Analytical performance model

As discussed in Section 1, we focus here on *update response time* as the main performance criterion for P2P ECA rule execution:

Definition 7. The *update response time* is the mean time required to complete all rule execution resulting from a single update submitted by a top-level transaction.

We do not attempt to capture the effects of parallel execution in the P2P network within this measure. The asynchronous nature of our P2P rule execution model makes it hard to capture parallelisation within an analytical model. Thus, the cost formulae that we derive for our analytical model can be considered as expressing the worst-case performance of P2P ECA rule execution.

The aim of our analytical model is to serve as a tool for studying the performance and scalability of P2P ECA rule processing systems. In designing the model, we have intentionally made some simplifying assumptions in order to prevent it from becoming overly complex. In our validation of the analytical model by means of a simulator for P2P ECA rule processing (which we discuss in more detail in Section 5.2), we relax most of these assumptions. The experimental results from the simulation experiments show good agreement with those predicted by our analytical model, hence confirming that we have indeed achieved a practically useful level of sophistication in the analytical model. Extending the analytical model in order to relax some of the assumptions made, and determining the impact of these changes, could be explored in further work.

We discuss the simplifying assumptions made for the analytical model next, and we also mention how some of these assumptions are relaxed in our simulator. More details of the latter will be discussed in Section 5.2. We stress that the ECA rule execution behaviour we described in Section 3, the analytical model we develop here, and the simulator that we describe in Section 5.2 are all independent of any specific data model, query language or update language.

4.1. Network and rule assumptions

Our model assumes a fixed number of peers, superpeers and ECA rules during rule execution — it does not capture the possibility of peers leaving or joining the network, or of rules being dynamically added or deleted. In real P2P systems, any of these may of course happen as transactions and rules execute. Capturing such occurrences formally within an analytical performance model is not straightforward, and we leave it as an area of future investigation.

4.2. Homogeneity assumptions

In common with other distributed systems performance studies (e.g. [15]) we make a homogeneity assumption separately for peers and superpeers. This assumption implies the same workload, service capacity and amount of data at each peer, and likewise at each superpeer. For superpeers, the assumption also implies the same number of rules in their rule bases. In our simulator, we allow the workload to vary according to varying transaction arrival rates at peers and superpeers. The amount of data per site and number of rules per superpeer are also varied.

The size of the messages exchanged between network nodes is also fixed in our analytical model. In the simulator, different message sizes are simulated by adding a normally distributed time delay to any message transmission. We note from the discussion in Section 3 that in a real system messages may include data items affected by an update at a peer, event and condition queries, results from such queries, and instances of rules' actions parts.

In the analytical model we assume a balanced peer distribution amongst superpeers, with each peergroup having the same number of peers. Again, in the simulator this is relaxed and the number of peers assigned to superpeers is randomly generated about a mean value.

4.3. Rule triggering assumptions

Transactions are submitted by applications to peers or superpeers in the network. Updates within these transactions are regarded as occuring at level 0 and they may trigger some rules. We assume that all rules have the same probability that they may be triggered by a given update — capturing more sophisticated probability distributions could be explored in further work. Some of the triggered rules may fire. Updates that result from the execution of these rules are regarded as occuring at level 1. Generally, the updates occurring at level $i+1$, $i \geqslant 0$, result from rules that have been fired by updates occurring at level i.

Definition 8. Let:

- n_{rules} denote the number of rules in each superpeer's rule base;
- p_{mt} denote the probability that a given rule may be triggered by a given update;
- r_{mt} denote the number of rules that may be triggered by a given update;
- $p_t(i)$, $i \geqslant 0$, denote the probability that a rule that may be triggered by an update occuring at level i is actually triggered;
- $r_t(i)$ denote the number of rules that are actually triggered by an update occuring at level i;
- p_f denote the probability that a rule that has been triggered fires;
- $p_{fire}(i)$ denote the probability that an update occurring at level i causes a given ECA rule to fire;
- $r_{fire}(i)$ denote the number of rules that fire after an update that occurs at level i.

In our experience, the level of ECA rule triggering in reactive applications tends to be shallow, in that the probability of having k levels of triggering in response to some top-level update decreases substantially as k increases. In particular, the applications discussed in Sections 1 and 2 exhibit this characteristic. Thus, for the analytical model, we assume that the probability $p_t(i)$ follows a *geometric distribution* with a constant reduction factor of p_{reduct} at each level, while p_{mt} and p_f remain constant regardless of the triggering level, i.e. that

$$p_t(i) = p_t \cdot p_{reduct}^i \tag{1}$$

for some value p_t.

We have also experimented with a constant reduction of the triggered probability by a factor of d_{reduct} at each level, so that $p_t(i)$ would be given by $p_t(i) = \max(0, p_t - i \cdot d_{reduct})$. We have found that this has no effect on the performance trends presented in Section 5, for either the analytical model or the simulator, except that the values of the update response time are observed to be higher than with a triggering probability that reduces geometrically.

It is easy to see that:

$$p_{fire}(i) = p_{mt} \cdot p_t(i) \cdot p_f \tag{2}$$

The following is a consequence of Eqs. (1) and (2):

$$p_{fire}(i) = p_{mt} \cdot p_t \cdot p_f \cdot p_{reduct}^i \tag{3}$$

The number of rules that may be triggered by an update (at any level i) is given by:

$$r_{mt} = p_{mt} \cdot n_{rules} \tag{4}$$

We therefore have the following expressions for the number of rules that are actually triggered by an update occurring at level i, $r_t(i)$, and the number of rules that fire for an update occurring at level i, $r_{fire}(i)$:

$$r_t(i) = p_t(i) \cdot r_{mt} = p_t \cdot p_{reduct}^i \cdot p_{mt} \cdot n_{rules} \tag{5}$$

$$\begin{aligned} r_{fire}(i) &= p_{fire}(i) \cdot n_{rules} \\ &= p_{mt} \cdot p_t \cdot p_{reduct}^i \cdot p_f \cdot n_{rules} \end{aligned} \tag{6}$$

4.4. System modelling

Our analytical model includes two kinds of queues: a *transaction queue* at each peer and superpeer that accepts queries or updates arising from rule execution, and an *action scheduler queue* at each superpeer that queues transactions resulting from rule firing, which are then dispatched for execution to the appropriate transaction queues in the peergroup.

For each queue we assume a FCFS (First Come First Served) service discipline. The arrival rate is modelled as a Poisson process, i.e. the arrival of a new item does not depend on any previous item and the inter-arrival time is exponentially distributed. The exponential distribution of inter-arrival time leads to the *service time* also following an exponential distribution, where the service time is the time required for a query to be evaluated or an update to be executed at a peer or superpeer. An empirical justification of the exponential distribution of the service time is given in [15] and is as follows:

The time required for a query/update depends on the number of data items accessed. Queries/updates accessing a small number of data items are more frequent than those accessing a large number of data items. This leads to a geometrically distributed number of data items accessed per query/update. So the service time can be assumed to be exponential, which is the continuous version of geometrical.

Using Kendall's notation [9], the transaction queues and action scheduler queues are queues of the form $M/M/1$, where M indicates exponential distribution for both the process arrival rate and the service time, and 1 specifies a single service point for each queue.

We model the communication channel between superpeers, and between superpeers and peers, as a server with an infinite number of service points, introducing a delay to all messages depending on their size and the network bandwidth. The queue model description of this is $M/D/\infty$, where D indicates deterministic service time and ∞ an unlimited capacity network.[2]

Definition 9. Let $size_m$ denote the size in bits of each message (we recall that this is fixed in the analytical model); *bps* denote the network communication bandwidth between a peer and a superpeer, in bits/second; and *NET_FACTOR* denote how many times greater the communication bandwidth between a pair of superpeers is compared to the bandwidth between a peer and a superpeer. Let t_{delay}^{P-SP} denote the network communication delay for messages between a peer and a superpeer and t_{delay}^{SP-SP} for messages between two superpeers (we assume that latency is incorporated into these delays). We note that

$$t_{delay}^{P-SP} = \frac{size_m}{bps} \tag{7}$$

$$t_{delay}^{SP-SP} = \frac{t_{delay}^{P-SP}}{NET\_FACTOR} \tag{8}$$

4.5. Modelling update response time

We recall that the update response time is the mean time required to complete all rule execution resulting from a single top-level update submitted to a peer or superpeer. We denote the update response time by $\overline{R}_{update}$ and we decompose it into three components:

$$\overline{R}_{update} = \overline{R}_{event} + \overline{R}_{cond} + \overline{R}_{action} \tag{9}$$

where $\overline{R}_{event}$ is the mean time required for all event processing, $\overline{R}_{cond}$ the mean time required for all condition processing and $\overline{R}_{action}$ the mean time required for all action processing. The three propositions below present an expression for each of these in terms of a set of fundamental quantities. Unless stated to the contrary, in what follows the term 'peer' is taken to include superpeers as well.

Proposition 1. *Let N_{action} be the mean number of updates contributed to an action schedule by the action part of a rule*; *m the number of peers in a peergroup* (*including the supervising superpeer*); λ_{peer} *the arrival rate of queries or updates at each peer's transaction queue*; t_q *the time needed for an event query or an update to be evaluated at a peer, and also the time to evaluate the intersection of the results of an event query with the set of data items affected by an update* (*for simplicity, we assume the same cost for all three of these here, though three different quantities could be captured within the model relatively straightforwardly*). *Then $\overline{R}_{event}$ is given by*

$$\overline{R}_{event} = \sum_{i=1}^{k} \left(r_{fire}(i-1) \cdot N_{action} \cdot \left[t_{delay}^{P-SP} \cdot \frac{m-1}{m} + r_{mt}(i) \cdot \left(2 \cdot t_{delay}^{P-SP} \cdot \frac{m-1}{m} + t_{peer\_total} + t_q \right) \right] \right)$$

where

$$t_{peer\_total} = t_q + \frac{\lambda_{peer} \cdot t_q^2}{1 - \lambda_{peer} \cdot t_q} \tag{10}$$

Proof. Let $\overline{R}_{event}(i)$ denote the mean time required for all event processing at level i. Then,

$$\overline{R}_{event}(i) = r_{fire}(i-1) \cdot N_{action} \cdot \left(\overline{T}_{event}^{db}(i) + \overline{T}_{event}^{net}(i) \right) \tag{11}$$

where $r_{fire}(i-1) \cdot N_{action}$ is the number of updates caused by the previous level of rule firing, and $\overline{T}_{event}^{net}(i)$ and $\overline{T}_{event}^{db}(i)$ denote the mean times spent in network processing and in query processing, respectively, during the evaluation of rule event queries following an update at level i.

[2] A more sophisticated but more complex way to model the communication would be to assume that each communication channel is a queuing network of $M/M/1$ queues.

When a level i update occurs at a peer P, P transmits a message to its coordinating superpeer SP. SP determines which rules in its rule base may be triggered by the update. The event query of each of the $r_{mt}(i)$ rules that may be triggered is sent by SP to P for evaluation. The evaluation results for each event query are transmitted back to SP, which matches them against the set of data items affected by the update. If the intersection of the two sets is non-empty, then the rule is actually triggered.

The network processing time of the above process is

$$\overline{T}^{net}_{event}(i) = t^{P-SP}_{delay} \cdot \frac{m-1}{m} + 2 \cdot t^{P-SP}_{delay} \cdot \frac{m-1}{m} \cdot r_{mt}(i) \tag{12}$$

where $\frac{m-1}{m}$ is the probability that the update has not occurred at a superpeer (i.e. that P is distinct from SP) and $t^{P-SP}_{delay} \cdot \frac{m-1}{m}$ is the time required for the transmission of a message between a peer and its superpeer. If the update has occurred at a superpeer, then no message transmission is necessary.

The query processing time of the above process is

$$\overline{T}^{db}_{event}(i) = r_{mt}(i) \cdot t_{peer\_total} + r_{mt}(i) \cdot t_q \tag{13}$$

Here, t_q is the mean time required to evaluate the intersection of the results of an event query with the set of data items affected by an update, and t_{peer_total} comprises the mean time, t_q, needed for a rule's event query to be evaluated at a peer plus the mean time, $\overline{W}_{peer}$, spent by query or update waiting in the peer's transaction queue. $\overline{W}_{peer}$ is given by the following, since the transaction queue is $M/M/1$ [9]:

$$\overline{W}_{peer} = \frac{\lambda_{peer} \cdot t_q^2}{1 - \lambda_{peer} \cdot t_q}$$

We thus obtain the definition for t_{peer_total} given in Eq. (10). Substituting Eqs. (12) and (13) into Eq. (11) and summing over k we obtain the expression stated in the proposition. □

Definition 10. We define a query *step* to be the smallest valid subquery of a query. We denote by $p_{annot_{cond}}$ the probability that a rule condition is c-relevant to one of the peers of a given peergroup, i.e. the probability that at least one of the steps within the condition's queries can be evaluated at the peer. Thus,

$$p_{annot_{cond}} = 1 - (1 - p_s)^{n_{steps}}$$

where n_{steps} denotes the mean number of steps within a rule's condition and p_s the probability that a given query step can be evaluated at a given peer.

We note that the probability p_s is also indicative of the amount of schema and data replication between peers in the network: a higher value of p_s implies greater replication since it represents a higher probability that a given query step can be evaluated at a given peer. Since a query step may in general include references to both schema objects (tables, attributes etc.) and specific data values, the probability p_s captures both schema and data replication. A value of $p_s = 1$ corresponds to full schema and data replication at all peers. In the analytical model p_s cannot equal 0 as this would be tantamount to stating that there is no data in the network.

Proposition 2. *Let N_{cond} be the mean number of instances generated for a rule's condition part when determining if the rule has fired (recall that set-oriented rules result in one instance and instance-oriented rules in multiple instances). Then $\overline{R}_{cond}$ is given by*

$$\overline{R}_{cond} = r_{mt} \cdot N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot \left(t_{peer\_total} \cdot m + 2 \cdot t^{P-SP}_{delay} \cdot (m-1)\right) \cdot p_t \cdot \frac{1 - p^{k+1}_{reduct}}{1 - p_{reduct}} \tag{14}$$

Proof. From the set of may-be-triggered rules, only a subset is actually triggered. The number of rules $r_t(i)$ that are actually triggered at level i is given by Eq. (5). For each of these rules, the time $\overline{R}^{rule}_{cond}$ needed for its condition part to be evaluated is given by

$$\overline{R}^{rule}_{cond} = \overline{T}^{db}_{cond} + \overline{T}^{net}_{cond} \tag{15}$$

where $\overline{T}^{db}_{cond}$ is the time spent on query processing and $\overline{T}^{net}_{cond}$ the time spent on network transmission.

Evaluating the condition part of a rule involves generating N_{cond} instances of it, determining to which of the peers in the peergroup subqueries of the condition should be sent for evaluation, sending the subqueries to these peers, evaluating each subquery at each peer, and finally sending the subquery results back to the superpeer. Recall that each of the N_{cond} instances has a mean number of n_{steps} steps within its condition. Thus

$$\overline{T}^{db}_{cond} = t_{peer\_total} \cdot N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot m \tag{16}$$

and

$$\overline{T}^{net}_{cond} = 2 \cdot t^{P-SP}_{delay} \cdot N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot (m-1) \tag{17}$$

where $m-1$ is the number of peers to which each message is sent (excluding the superpeer).

From Eqs. (15), (16) and (17) we obtain the mean time needed for the evaluation of the condition part of a single rule:

$$\overline{R}^{rule}_{cond} = N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot \left[t_{peer\_total} \cdot m + 2 \cdot t^{P-SP}_{delay} \cdot (m-1)\right] \tag{18}$$

So the mean time required to process all condition queries over all k levels during the rule execution that follows a top-level update is:

$$\begin{aligned}\overline{R}_{cond} &= \sum_{i=0}^{k} \left(r_t(i) \cdot \overline{R}^{rule}_{cond}\right) \\ &= r_{mt} \cdot N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot \left(t_{peer\_total} \cdot m + 2 \cdot t^{P-SP}_{delay} \cdot (m-1)\right) \cdot p_t \cdot \sum_{i=0}^{k} p^i_{reduct}\end{aligned} \tag{19}$$

which further simplifies to the expression stated in the proposition. □

Definition 11. Let $p_{annot_{action}}$ denote the probability that all the updates within an instance of a rule's action part can be executed at a given peer, i.e.

$$p_{annot_{action}} = p_s^{n_{steps}} \tag{20}$$

where, similarly to rule conditions, p_s expresses the probability that a given update can be evaluated at a given peer (for simplicity, we assume the same number of updates, n_{steps}, in a rule's action part as there are steps in its condition part – the model could easily be refined to capture two different quantities). Let λ_{ext} be the arrival rate of top-level updates submitted from outside the rule processing system.

Lemma 3. *The mean time spent within a peergroup on the execution of an update at level $i > 0$ is given by*

$$T_{update\_exec}(i) = t_{srv} + \frac{\lambda_{total}(i) \cdot t^2_{srv}}{1 - \lambda_{total}(i) \cdot t_{srv}} \tag{21}$$

where $t_{srv} = p_{annot_{action}} \cdot m \cdot t_q$ and

$$\lambda_{total}(i) = \lambda_{ext} \cdot \left(1 + n_{rules} \cdot p_{mt} \cdot p_t \cdot p_f \cdot \sum_{j=1}^{i} N^j_{action} \cdot \prod_{l=1}^{j} p^l_{reduct}\right) \tag{22}$$

Proof. After an instance of the action part is placed on an action scheduling queue at level $i > 0$ of rule execution, each update within it waits for a time $\overline{W}(i)$ before being sent, as part of the whole instance, to the appropriate peers of the peergroup where it receives a total service time of t_{srv}, i.e.

$$T_{update\_exec}(i) = t_{srv} + \overline{W}(i) \tag{23}$$

Since an update will be executed on $(p_{annot_{action}} \cdot m)$ peers of the peergroup, t_{srv} is given by

$$t_{srv} = p_{annot_{action}} \cdot m \cdot t_q$$

The mean waiting time in the $M/M/1$ action scheduling queue, at level i, is given by the following [9]:

$$\overline{W}(i) = \frac{\lambda_{total}(i) \cdot t^2_{srv}}{1 - \lambda_{total}(i) \cdot t_{srv}} \tag{24}$$

Here, $\lambda_{total}(i)$ is the total arrival rate of updates at a peergroup at level i:

$$\lambda_{total}(i) = \lambda_{ext} + \lambda_{int}(i) \tag{25}$$

where λ_{ext} is the arrival rate of top-level updates and $\lambda_{int}(i)$ is the arrival rate of updates generated as a result of rules firing. We obtain an expression for $\lambda_{int}(i)$ in terms of λ_{ext} by observing that each top-level update causes $r_{fire}(1)$ rules to fire, each of which generates N_{action} updates at level 1; each of these updates causes $r_{fire}(2)$ rules to fire, each of which generates N_{action} updates at level 2; and so forth up to level i. So the total arrival rate of updates at a peergroup at level i is:

$$\begin{aligned}\lambda_{total}(i) &= \lambda_{ext} + \lambda_{ext}\cdot r_{fire}(1)\cdot N_{action} + \lambda_{ext}\cdot r_{fire}(1)\cdot r_{fire}(2)\cdot N_{action}^2 \\ &\quad + \cdots \\ &\quad + \lambda_{ext}\cdot r_{fire}(1)\cdots r_{fire}(i-1)\cdot r_{fire}(i)\cdot N_{action}^i \\ &= \lambda_{ext}\cdot\left(1+\sum_{j=1}^{i} N_{action}^j\cdot\prod_{l=1}^{j} r_{fire}(l)\right) \end{aligned} \tag{26}$$

From Eqs. (6) and (26), the transaction arrival rate at level i is therefore as given in Eq. (22). □

Proposition 4. *Let n be the number of superpeers; n_{hops} be the mean number of message transmission steps (hops) required for an instance of a rule's action part to be transmitted to all superpeers (this depends on the network topology and the routing policy); and p_{allow} be the probability that a superpeer has within its peergroup the schema and data resources necessary to execute the instance, and that its data access privileges allow access to these resources. Then the mean time required for all action processing, $\overline{R}_{action}$, is given by*

$$\overline{R}_{action} = \sum_{i=1}^{k} r_{fire}(i)\cdot\left(\overline{T}_{action}^{db}(i) + \overline{T}_{action}^{net}(i)\right) \tag{27}$$

where

$$\overline{T}_{action}^{net}(i) = \left[t_{delay}^{P-SP}\cdot p_{annot_{action}}\cdot(m-1) + \left(t_{delay}^{SP-SP}\cdot n_{hops} + t_{delay}^{P-SP}\cdot(n-1)\cdot p_{allow}\cdot p_{annot_{action}}\cdot(m-1)\right)\right]\cdot N_{action} \tag{28}$$

and

$$\overline{T}_{action}^{db}(i) = N_{action}\cdot T_{update\_exec}(i)\cdot\left(1+(n-1)\cdot p_{allow}\right) \tag{29}$$

Proof. The mean time needed for the execution of the action part of a rule at level i, $\overline{R}_{action}^{rule}(i)$, is the sum of the time spent on network transmission and the time spent on query evaluation and update execution at peers, which we denote by $\overline{T}_{action}^{net}(i)$ and $\overline{T}_{action}^{db}(i)$, respectively:

$$\overline{R}_{action}^{rule}(i) = \overline{T}_{action}^{net}(i) + \overline{T}_{action}^{db}(i) \tag{30}$$

Each instance of the action part of the rule (one instance in the case of a set-oriented rule, one or more instances in the case of an instance-oriented rule) is sent to the peers of the local peergroup according to the annotations on the rule's action part. This takes a time of $t_{delay}^{P-SP}\cdot p_{annot_{action}}\cdot(m-1)$ for each update within the rule's action part. Each instance of the action part of the rule is also sent to all other $n-1$ superpeers of the network where it will be scheduled for execution depending on the probability p_{allow}. If the execution is possible, the updates comprising the instance of the action part will be sent to $p_{annot_{action}}\cdot(m-1)$ peers of the peergroup (excluding the superpeer itself). This takes a time of $t_{delay}^{SP-SP}\cdot n_{hops} + t_{delay}^{P-SP}\cdot(n-1)\cdot p_{allow}\cdot p_{annot_{action}}\cdot(m-1)$ for each update. Summing over the N_{action} updates within an instance of the action part gives Eq. (28).

The execution time for the updates within the local peergroup is $N_{action}\cdot T_{update\_exec}(i)$ where $T_{update\_exec}(i)$ is as defined in the previous lemma. The time needed for all instances of a rule's action part to be executed on the $n-1$ remote peergroups is the same, but multiplied by the probability p_{allow}. So the total time needed for the execution of the action part of a rule at level i is given by Eq. (29).

Summing over the number of rules that fire at level i and over all k levels gives the time required for all rule action processing given in Eq. (27). □

5. Experimental results

We have developed a simulator of an archetypal P2P ECA rule processing system, in order to validate the predictions of our analytical model. We have conducted experiments with both the analytical model and the simulator for two different network topologies, random and HyperCup [19]; we note here that the network topology relates to the communication between superpeers, since each (non-superpeer) peer has only one active network connection with its supervising superpeer. We explore below the update response time with each topology, for varying values of n, n_{rules} and p_s. The results of our experiments are discussed in Section 5.1 and Section 5.2.

For the experiments with the analytical model, the values of the rule-related parameters N_{cond}, N_{action}, p_{reduct}, p_{allow}, p_t, p_f, p_{mt} and n_{steps} are set as shown in Table 1. In the absence of more general information about ECA rule sets for P2P applications, we based these values on the types of ECA rules and data anticipated by the SeLeNe e-learning application [12].

Table 1
Parameter base values.

Parameter	Base setting	Parameter	Base setting
λ_{ext}	20 trans/s	λ_{peer}	5 trans/s
bps	512 kbits/s	*NET_FACTOR*	20
$size_m$	10 kbytes	t_q	1 s
k	30	N_{cond}	4
N_{action}	8	p_{reduct}	0.2
p_{allow}	0.9	p_t	0.5
p_f	0.5	p_{mt}	0.2
n_{steps}	4	m	20

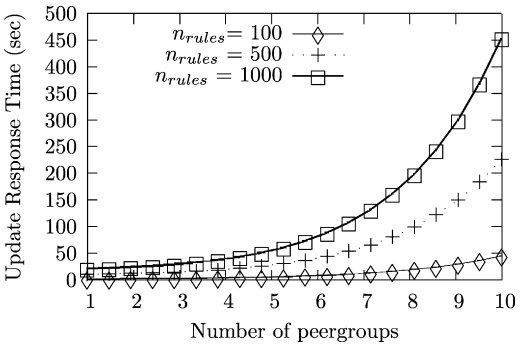

Fig. 1. Analytical model, data replication 10%, random topology.

Chapter 6 of [17] lists an indicative set of such rules. We also used this application to determine $size_m$ and t_q in Table 1 by running a set of sample queries and updates over the RDF repository underlying RDFTL. We have adopted a value of k that is of the same order of magnitude as adopted in commercial DBMSs as an upper limit for the number of recursive rule firings allowed. The value of N_{action} is relatively low because "bulk" updates are likely to be infrequent in the applications discussed in Section 1. We have experimented with different settings of the above parameters and this affects only the absolute response time values and not the performance trends of the graphs we present later.

The values of *NET_FACTOR* and *bps* are based on speeds of 10 Mbps and 512 kbps in the superpeer-to-superpeer and peer-to-superpeer connections, respectively. Again, we have experimented with higher and lower values of *NET_FACTOR*, both in the analytical model and the simulator. These shift the update response time values up or down depending on the chosen value, but have no effect on the performance trends.

The number of peers in each peergroup is set at $m = 20$ in the analytical model, the arrival rate of top-level updates at 20 trans/s and the transaction arrival rate at peers at 5 trans/s. Again, altering these values changes the absolute response time values in the analytical model and the simulator, but has no effect on the performance trends.

5.1. Analytical study results

We examine for each of the two network topologies (random and HyperCup [19]) the update response time with respect to varying numbers of peergroups in the network (n), varying numbers of rules per rule base (n_{rules}) and varying data replication (p_s). In the first topology, the superpeers are assumed to be connected at random. Any message between them is broadcast from the originating superpeer to all its neighbouring superpeers, and from there to all their neighbours, and so forth, flooding the network until the message reaches all the superpeers. This simple topology does not place any guaranteed upper bound on the number of hops that will be necessary for a message to reach all superpeers. It also does not prevent a message being received more than once by the same superpeer.

Fig. 1 shows how the update response time varies with this topology as the number of peergroups n increases, with $p_s = 0.1$ and the number of rules per rule base n_{rules} set to 100, 500 and 1000. From the shape of these curves it is clear that the system does not scale well. Similar experiments conducted for higher values of p_s result in graphs with similar upwards trends, except that the values of the update response time are larger and the system becomes unstable at lower values of n. This is to be expected as the presence of data in more peergroups increases the number of peers that a rule which has fired can be executed on, thus increasing the network traffic, the load on peers, and the overall number of rule firings.

Using instead the HyperCup topology guarantees that each superpeer receives a message only once. A total of $n-1$ hops are required for a message to reach all superpeers. The most distant superpeers can be reached in $\log_2 n$ hops. Fig. 2 shows how the update response time varies as n increases, with the data replication p_s being set to 0.1 and the number of rules per rule base, n_{rules}, set to 100, 500 and 1000. We see that the system now shows good scalability, and the update response time increases linearly with n. Compared with the random topology, the system remains stable and the update response

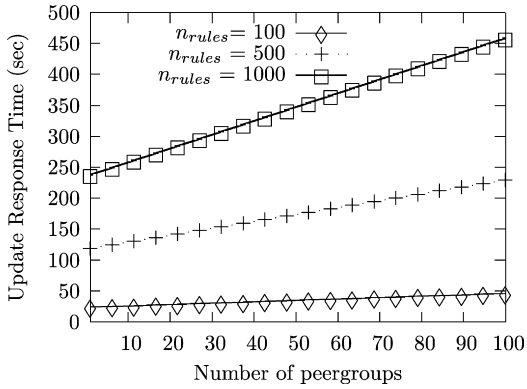

Fig. 2. Analytical model, data replication 10%, HyperCup topology.

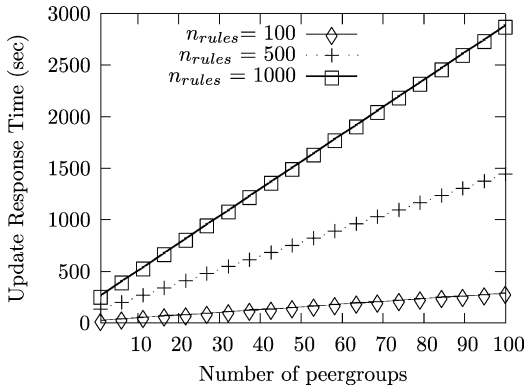

Fig. 3. Analytical model, data replication 90%, HyperCup topology.

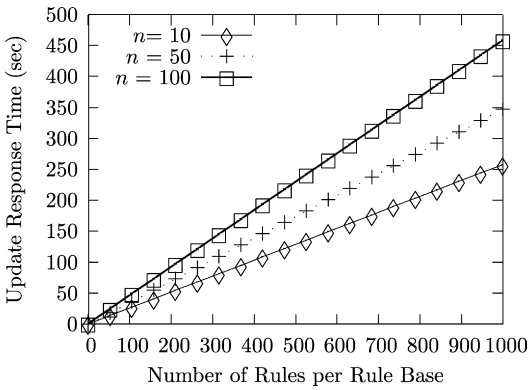

Fig. 4. Varying no. of rules; analytical model, data replication 10%, HyperCup topology.

time within reasonable boundaries (we note that in popular P2P systems it sometimes takes minutes for a search request to complete).

As with the random topology, increasing the data replication, p_s, increases the average update response time. However, in the case of HyperCup, the system still remains stable and the update response time still increases linearly even for high values of p_s, e.g. Fig. 3 illustrates the case of $p_s = 0.9$. Fig. 4 shows how the update response time varies as the number of rules per rule base, n_{rules}, increases, with the data replication p_s being set to 0.1 and the number of peergroups set to 10, 50 and 100. We see that the system again shows good scalability, with the update response time increasing linearly with n_{rules}.

For the analytical model, we have used values up to $n = 10{,}000$ and $n_{rules} = 10{,}000$, with varying values of p_s, and the same linear trends are observed with the HyperCup topology as discussed above. Similar experiments have been conducted using different settings for the parameters of Table 1, and this affects only the absolute performance values, and not the performance trends, with two exceptions: (i) Increasing p_{reduct} (the reduction factor at each level of the 'may-be-triggered' probability) up to approximately 0.5 does not affect the performance trends. However, for values above 0.5, it gives similar

performance trends only for values of p_s of up to about 0.4. For higher p_s, the update response time becomes non-linear as the value of n_{rules} or n increases. This can be explained by the fact that higher data replication results in more rules being triggered, more instances of rule actions being sent over the network and hence an increase in network transmission times. Moreover, the arrival rate of updates becomes higher than the rate at which they can be served, causing significant increases in the queue waiting times. (ii) Varying p_{mt} (the probability that a given rule may be triggered by a given update) we observe that with p_{mt} values up to 0.3 the system shows good scalability and the update response time increases linearly. As the value of p_{mt} increases above 0.3, the system shows a non-linear behaviour that becomes progressively more severe.

5.2. Simulation results

We have developed a simulator of a P2P ECA rule processing system in order to validate the predictions of our analytical model. This simulator was built using the Java implementation of the SSF (Scalable Simulation Framework) API,[3] called Raceway SSF,[4] The main entities of our simulator are the peers, superpeers, rulebases and rules. Like the analytical model, our simulator is independent of any specific data model and query/update language. When the simulator initialises, it creates a pre-specified number of superpeer objects, and "connects" to each of them a number of peer objects. Which other superpeer objects each superpeer object is connected to depends on the network topology. Parameters such as the network delay, number of rules in a rule base and number of actions per rule are represented by variables in the relevant object. A number of statistical distributions are used to simulate the behaviour of the varying parameters (we discuss these below). Each time a value of one of these parameters is needed, a random number generator generates a new random value according to the given distribution.

Our simulator generates an Event Handler, Condition Evaluator and Action Scheduler object for each superpeer object, and these are collectively responsible for simulating the P2P rule execution logic described in Section 3.1. In particular, the Event Handler is notified of updates by peers in its peergroup, and determines which rules have been triggered by an update at a peer by invoking that peer's query service to evaluate the event queries of rules that may have been triggered. The Condition Evaluator determines which of the triggered rules should fire, by generating the appropriate condition instances, and coordinating their evaluation across the peergroup. The Action Scheduler generates from the action parts of rules that have fired a list of updates, and coordinates their execution across the peergroup as well as sending them to all other superpeers. The peer data services are simulated by two functions, one for updates and one for queries. Calling each of these functions adds a time delay (corresponding to the query or update processing time) derived by calling a random number generator to produce numbers about the mean value. The messaging service is simulated by a function that adds a time delay, corresponding to network transmission time, again produced by a random number generator about the mean value. The rule base management services are simulated by functions in the superpeer class, while the rule base is simulated by a class of its own. The rule base class has as its main parameter the number of rules in this superpeer's rulebase, and contains methods for setting and retrieving the number of rules. The routing service at each superpeer is simulated by a look-up into the superpeer's list of connections to other superpeer and peer objects.

A specified number of external transactions is executed per simulation run, randomly submitted to the peer and superpeer objects. An internal clock, initialised and managed by the simulation application, records the number of time units required for executing each function of the simulation application.

The same set of experiments as for the analytical model were performed with the simulator. The same values for *bps*, *NET_ FACTOR*, k, p_{reduct} and p_{mt} were used as for the analytical model experiments. The remaining system parameters of Table 1 were replaced by stochastic values. In particular, the number of peers per peergroup, m, is randomly distributed with a mean value of 20; the probabilities p_{allow}, p_t and p_f are normally distributed, with mean values as in Table 1; and the λ_{ext}, λ_{peer}, $size_m$, t_q, N_{cond} and N_{action} parameters are exponentially distributed, with mean values as in Table 1. Each of the data points in the graphs was obtained by running the simulator for 2000 top-level updates and taking the average update response time. The experiments were again conducted with both the random and the HyperCup topologies. In the experiments with the simulator, the number of rules per rule base, n_{rules}, and the degree of replication, p_s were not fixed at the chosen values, but normally distributed about these chosen values.

Fig. 5 shows how the update response time varies with the random topology as the number of peergroups, n, increases, with p_s distributed about a mean value of 0.1 and n_{rules} distributed about mean values of 100, 500 and 1000. We see that the trend of these graphs is similar to Fig. 1 from the analytical model. Experiments conducted with the simulator for higher average values of p_s result in graphs with similar upwards trends as with the analytical model.

Some results from the same set of experiments using a HyperCup topology are shown in Figs. 6 and 7. We see that the system now shows good scalability, similarly to the corresponding results from the analytical model in Figs. 2 and 3. Fig. 8 shows how the update response time varies as n_{rules} increases, with p_s being distributed around a mean value of 0.1, and the number of peergroups set to 10, 50 and 100. We see that the system shows good scalability, with the update response time increasing linearly with n_{rules}, similarly to the corresponding results from the analytical model study in Fig. 4.

[3] http://www.ssfnet.org/homePage.html.

[4] https://gradus.renesys.com/exe/Raceway.

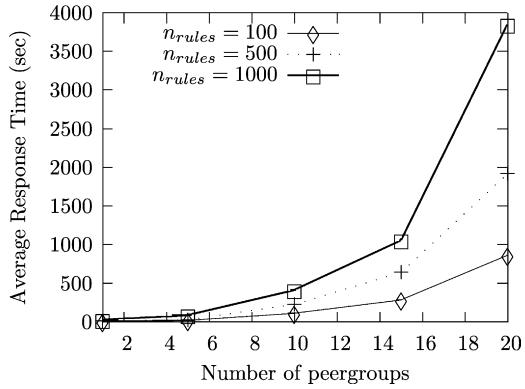

Fig. 5. Simulation, data replication 10%, random topology.

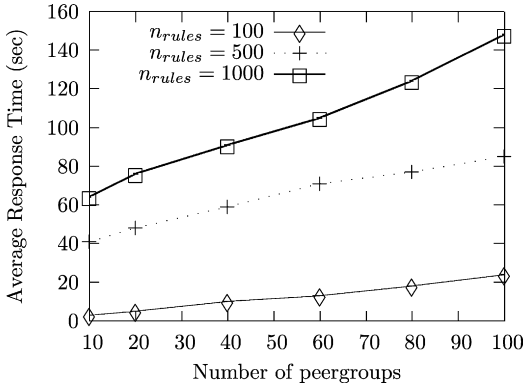

Fig. 6. Simulation, data replication 10%, HyperCup topology.

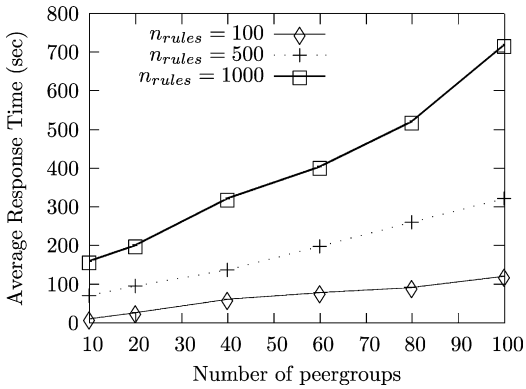

Fig. 7. Simulation, data replication 90%, HyperCup topology.

The difference in the absolute values of the update response time between the analytical model and the simulations can be explained by the fact that in the analytical model we have used fixed values for the system parameters of Table 1 while in the simulations the values of these parameters vary following some distribution. The absolute values of the results from the simulations are likely to be closer to real systems, but their trends show good agreement with those of the analytical model.

6. Conclusions and future work

We have developed an analytical model for studying the performance of processing ECA rules in schema-based P2P systems. We have examined predictions from the analytical model regarding how update response time varies with the network topology, number of peers, number of rules and degree of schema and data replication between peers. We have also described a simulation of an archetypal P2P ECA rule processing system, and have conducted similar experiments with

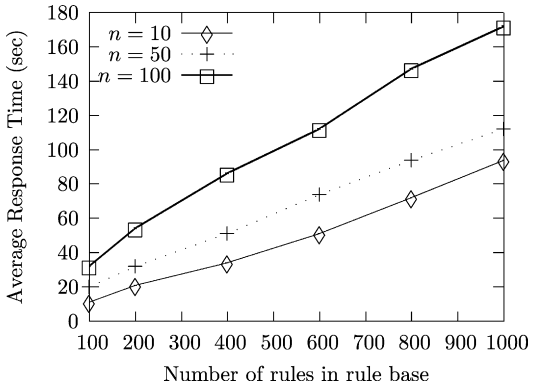

Fig. 8. Varying no. of rules; simulation, data replication 10%, HyperCup topology.

the simulator. The two sets of experimental results show good agreement, which is an indication of the validity of our analytical model.

To our knowledge, this is the first time that the performance of P2P ECA rule processing has been studied using analytical and simulation methods. Both sets of experiments show that performance is reliant on the network topology between superpeers, and we have seen that ECA rule processing shows good scalability if a HyperCup topology is used to connect the superpeers.

The main contributions of this paper are (i) the analytical model and (ii) the results of the performance study undertaken using this model, which have been confirmed by the simulation results. Our analytical model aims to serve as a general tool for studying the performance and scalability of P2P ECA rule processing systems. As such, the model is independent of any specific data model, query language or update language. It would also be straightforward to apply our modelling and simulation approaches to 'pure' schema-based P2P systems, where there are no 'super' peers and, in effect, all peers have superpeer capabilities.

In our P2P ECA rule execution model, each instance of a rule's actions part consists of updates that are executed at a single peer, and rules' condition queries are assumed to be evaluated locally within a single peergroup. Extending our analytical model to capture global condition evaluation using global P2P query processing techniques would be relatively straightforward, as would capturing more expressive rule actions consisting of possibly distributed queries and updates.

We have focussed so far on response time as the main performance criterion because this seems to be the major concern in the applications we have identified. However, similar analytical models could be defined, and simulation experiments conducted, to study various types of resource consumption, e.g. CPU, I/O and network usage. Other future work would be to extend our analytical model to capture composite events, to handle additional rule coupling modes, to capture more sophisticated rule triggering probability distributions, and to relax some of the assumptions made in Section 4 and determine the impact of these changes on the analytical model's predictive accuracy.

References

[1] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, T. Milo, Dynamic XML documents with distribution and replication, in: Proc. SIGMOD, 2003, pp. 527–538.
[2] A. Aiken, J. Widom, J.M. Hellerstein, Static analysis techniques for predicting the behavior of active database rules, ACM Trans. Database Systems 20 (1) (1995) 3–41.
[3] E. Baralis, A. Bianco, Performance evaluation of rule semantics in active databases, in: Proc. ICDE'97, 1997, pp. 365–374.
[4] E. Baralis, S. Ceri, S. Paraboschi, Compile-time and runtime analysis of active behaviors, IEEE Trans. Knowl. Data Engrg. 10 (3) (1998) 353–370.
[5] E. Baralis, J. Widom, An algebraic approach to static analysis of active database rules, ACM Trans. Database Systems 25 (3) (2000) 269–332.
[6] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, I. Zaihrayeu, Data management for peer-to-peer computing: a vision, in: Proc. WebDB'02, 2002, pp. 89–94.
[7] P.-A. Chirita, S. Idreos, M. Koubarakis, W. Nejdl, Publish/subscribe for RDF-based P2P networks, in: Proc. First European Semantic Web Symposium, 2004, pp. 182–197.
[8] A. Geppert, S. Gatziu, K.R. Dittrich, A designer's benchmark for active database management systems: 007 meets the BEAST, in: Proc. 2nd Int. Workshop on Rules in Database Systems, 1995, pp. 309–326.
[9] R. Jain, The Art of Computer Systems Performance Analysis, Wiley, 1991.
[10] V. Kantere, I. Kiringa, J. Mylopoulos, A. Kemenstiestides, M. Arenas, Coordinating peer databases using ECA rules, in: Proc. DBISP2P, 2003, pp. 108–133.
[11] V. Kantere, A. Tsois, Using ECA rules to implement mobile query agents for fast-evolving pure P2P database systems, in: Proc. 6th Int. Conf. on Mobile Data Management, 2005, pp. 164–172.
[12] K. Keenoy, A. Poulovassilis, V. Christophides, P. Rigaux, G. Papamarkos, A. Magkanaraki, M. Stratakis, N. Spyratos, P.T. Wood, Personalisation services for self e-learning networks, in: Proc. 4th Int. Conference on Web Engineering, 2004, pp. 215–219.
[13] A. Löser, M. Wolpers, W. Siberski, W. Nejdl, Efficient data store and discovery in a scientific P2P network, in: Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data (co-located with ISWC'03), 2003, at ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-83/scin_1.pdf.
[14] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, T. Risch, EDUTELLA: a P2P networking infrastructure based on RDF, in: Proc. WWW'2002, 2002, pp. 604–615.
[15] M. Nicola, M. Jarke, Performance modeling of distributed and replicated databases, IEEE Trans. Knowl. Data Engrg. 12 (4) (2000) 645–672.

[16] G. Papamarkos, A. Poulovassilis, P.T. Wood, Event-Condition-Action rules on RDF metadata in P2P environments, Comput. Networks 50 (10) (2006) 1513–1532.
[17] G. Papamarkos, Event-Condition-Action rule languages over semi-structured data, PhD thesis, Birkbeck, University of London, March 2007.
[18] N.W. Paton, Active Rules in Database Systems, Springer, 1999.
[19] M. Schlosser, M. Sintek, S. Decker, W. Nejdl, HyperCuP – hypercubes, ontologies and efficient search on P2P networks, in: Proc. 1st Int. Workshop on Agents and P2P Computing, 2002, pp. 112–124.
[20] W. Siberski, U. Thaden, A simulation framework for schema-based query routing in P2P networks, in: Proc. 1st Workshop on Peer-to-Peer Computing and Databases, 2004, pp. 436–445.
[21] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, P. Mork, The piazza peer data management project, SIGMOD Record 32 (3) (2003) 47–52.
[22] W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, A.P. Buchmann, A peer-to-peer approach to content-based publish/subscribe, in: Proc. 2nd Int. Workshop on Distributed Event-Based Systems, 2003, pp. 1–8.
[23] J. Widom, S. Ceri, Active Database Systems, Morgan Kaufmann, San Mateo, CA, 1995.
[24] M. Zoumboulakis, G. Roussos, A. Poulovassilis, Active rules for sensor databases, in: Int'l Workshop on Data Management for Sensor Networks (co-located with VLDB'04), 2004, pp. 98–103, 2003.